A Framework for Advancing Front Techniques of Finite Element Mesh Generation *

S Farestam †and R B Simpson ‡

September 7, 1994

Abstract

Advancing front techniques are a family of methods for finite element mesh generation that are particularly effective in dealing with complicated boundary geometries. In the first part of this paper, conditions are presented which ensure that any planar aft algorithm that meets these conditions terminates in a finite number of steps with a valid triangulation of the input domain. These conditions are described by specifying a framework of subtasks that can accommodate many aft methods and by prescribing the minimal requirements on each subtask that ensure correctness of an algorithm that conforms to the framework.

An important efficiency factor in implementing an aft is the data structure used to represent the unmeshed regions during the execution of the algorithm. In the second part of the paper, we discuss the use of the constrained Delaunay triangulation as an efficient abstract data structure for the unmeshed regions. We indicate how the correctness conditions of the first part of the paper can be met using this representation. In this case, we also discuss the additional requirements on the framework which ensure that the generated mesh is a constrained Delaunay triangulation for the original boundary.

Classifications AMS(MSC) 65N50 , 65Y25 ; CR G.1.8, I.3.5

Keywords unstructured meshes, finite element method, Delaunay triangulation

1 Introduction

Advancing front techniques are a family of closely related heuristic mesh generation methods for the finite element method particularly suited for domains with complicated boundary curves and internal interfaces. The name refers to the strategy of generating triangles sequentially from an ever shrinking set of dynamic curves that start at the boundaries and

*The first author has been supported by CERFACS, Toulouse, France. Support was provided to the second author by the Natural Sciences and Engineering Research Council of Canada, and by the Inf ormation Technology Research Centre of Ontario. The second author also enjoyed the hospitality of CERFACS during the collaboration on this research

†CERFACS, 42, Ave. Coriolis, 31057 , Toulouse, France

‡Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1

internal interfaces of the domain and advance into its interior, like the ice-liquid surface of a freezing ice-cube. These methods , like all mesh generation methods, balance considerations of correctness, execution efficiency, and suitability of the resulting triangular meshes to the application. (See Bern and Eppstein, [2] for a review of unstructured triangular meshes generally, and comments on advancing front techniques as heuristics.) In most of the literature on methods based on the advancing front technique (aft), the authors concentrate on the details that differentiate the particular version being presented from others in this family, by its contributions to efficiency and mesh quality.

In the first part of this paper, we present a foundation, or framework, which can accommodate most features of the aft family of methods by various specializations and which ensures that algorithms which adhere to this framework terminate in a finite number of steps having constructed a triangulation of the input domain, i.e. a framework for the correctness of aft methods. We are not aware of literature that addresses the correctness of a particular version of the aft. In fact, in George, [13], there is a discussion of an example of the failure of the aft method presented in that book. An algorithm is usually expected to be totally prescriptive of the computation that it is describing. However, the framework that we discuss only sets minimal requirements for correctness that can be met in a variety of ways to address goals of efficiency and mesh quality. In this sense it admits a variety of algorithms, so we refer to it as a polyalgorithm. The basic polyalgorithm for the advancing front technique is introduced in §2. An important issue for the discussion of the correctness of the polyalgorithm is a rigorous description of the geometry of an advancing front. We refer to this set of edges as a frontal edge set and describe its geometry in §2.1.

For the generation of unstructured triangular meshes on a polygonal domain, $\mathcal{D}$, two closely related strategies are common. In the Voronoi, (alternatively referred to as Delaunay triangulation) approaches, the nodes of the mesh are generated in $\mathcal{D}$ by some technique for suitably distributing them, and then appropriate triangle incidence connections are computed. Standard Delaunay triangulation algorithms compute a mesh on the convex hull of the node set to form triangles that are as close to equilateral as possible, in a well defined sense. Algorithms for this construct are well known and their correctness long established. An extensive review, including references to these algorithms, has been published by Aurenhammer, [1]. For finite element mesh generation, the mesh must conform to the typically non-convex boundary of $\mathcal{D}$, and possibly to some internal interfaces, so the classical Delaunay triangulation may be replaced for this task by the constrained Delaunay triangulation. Algorithms and correctness arguments for the constrained Delaunay triangulation have been published by Borgers, [3], Chew, [7], Cline and Renka, [9], Lee and Lin, [17], and Lo, [18]. Finite element mesh generation techniques based on this Voronoi approach have been presented by Chew, [8], Jian-Ming et al, [16] , and Vallet, Hecht, and Mantel, [29], and are reviewed in the book by George, [13], and in the thesis of M-G Vallet, [28].

In the aft approaches, the creation of mesh nodes is interspersed with the selection of triangle incidences for the mesh and the two tasks are directly coupled. I.e. the distinction between the Voronoi and aft approaches is primarily a methodological one concerned with the stage at which, and the mechanism by which, the nodes of the mesh are created. Various versions of the aft have been presented by Bykat, [5], Cabello et al, [6], Dannelongue and Tanguy, [10], Farestam, [11], Lohner and Parikh, [19], Peraire et al, [21], and Tilch, [26] , [27]. Our direct experience with aft algorithms that conform to the framework presented

here has been reported in [11], [26], and [27]. It appears to us that the methods discussed in the other references could conform to the framework; however, their descriptions give insufficient algorithmic detail to be certain.

Bui and Hanh, [4], and Lo, [18], illustrate well the close connection between the the aft and Voronoi approaches; in the methods presented in each of these papers, a node set in $\mathcal{D}$ is first generated , and then a triangulation constructed based on it. The authors refer to the construction of the triangles as being carried out by an advancing front technique, because the triangles are formed on an ever shrinking boundary-like strip. However, in our view these methods are probably better categorized as Voronoi approaches since the positions of the nodes are specified prior to the formation of the triangle incidences.

The second part of this paper discusses the constrained Delaunay triangulation as an efficient high level data representation for aft methods, and discusses how the minimal requirements of the polyalgorithm of §2 can be met using this data representation. The use of the constrained Delaunay triangulation as a data structure for the aft has recently been proposed by Müller, Roe and Deconinck, [20]. We then conclude with comments on how to extend the basic aft framework to ensure that the generated triangulation is the constrained Delaunay triangulation of its nodes and the original boundary curves.

Aft methods require efficient techniques for establishing the visibility in the unmeshed region of a vertex from an edge. Actually, any triangulation of the unmeshed region can support efficient visibility checking, as we indicate in §3, and as has been noted in different forms by Chew, [8], Guibas et al, [14], [20], and Lo, [18]. The use of the constrained Delaunay triangulation in particular has some additional efficiencies if it is desired that the mesh to be generated be the constrained Delaunay triangulation of its vertices and boundary and interface edges of $\mathcal{D}$. To underscore these distinctions, we could have elected to discuss the extensions of the polyalgorithm of §2 first to the use of arbitrary triangulations to support visibility checking, and then to the use of the constrained Delaunay triangulation. However, the resulting cross referencing of ideas in these two closely linked topics would seem overly tedious, so we combine the two discussions and leave it to the reader to note the distinctions in question.

2 The Basic Polyalgorithm

The polyalgorithm is given as the procedure **Basic_aft** in Figure 5. It has been broken down into subtasks with the usual modularity properties of having simpler, relatively independent substrategies and reduced data access requirements. Each subtask is discussed subsequently, specifying minimal requirements for it that will ensure a correct algorithm. These minimal requirements underspecify the subtasks; the remaining flexibility can be directed to efficiency and mesh quality goals. Some commentary on possible, or typical, strategies for these purposes is included but the subtasks are not required to use them. The minimal requirements ensure the correctness of the polyalgorithm in the sense that:

a) the minimal requirements are feasible

b) if subtasks are specified which conform to the minimal requirements, their use in the framework of the polyalgorithm results in an algorithm that is guaranteed to terminate

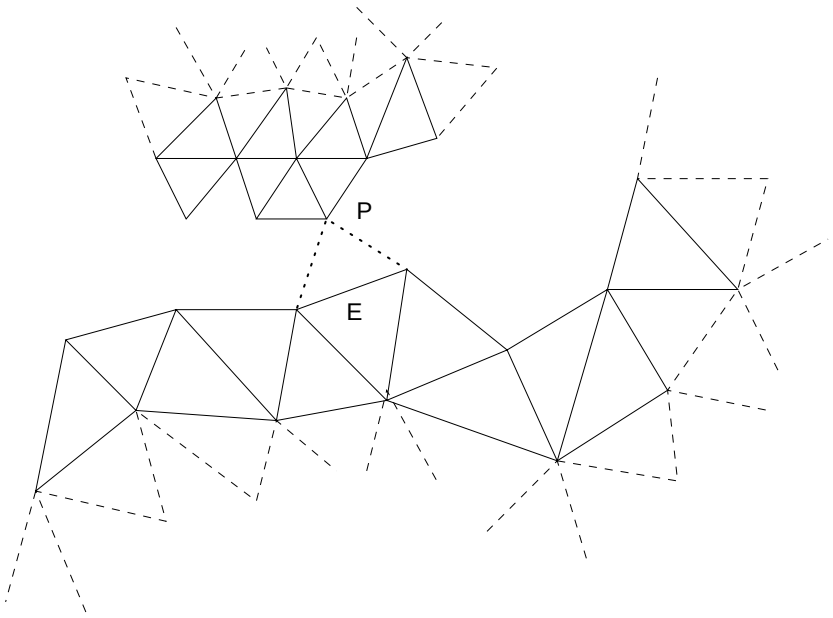

Figure 1: Triangle creation pinches off a multiply connected unmeshed subdomain

producing a triangulation of the input domain.

Naturally, subtask strategies which only meet the minimal requirements, while producing valid triangulations for complex geometries, will almost certainly yield aft methods that produce meshes of unacceptably low triangle quality or high execution cost.

Before discussing **Basic_aft**, however, we need to discuss the basic data structure that it operates on.

2.1 Boundary curves and frontal edge sets

Afts maintain dynamic lists of edges which form boundary curves, called fronts, for subsets of the domain that have not yet been triangulated. In this section we describe some geometric properties of these curves and domains, and give a formal description of this set of edges , which we will refer to as a frontal edge set. The complexity of an accurate description of these curves and domains is strongly influenced by the form of connectedness of the input domains to the aft that we chose to allow in our discussion. If we restrict the input domains to be the simply connected interiors of simple closed polygonal curves, then the least complicated exposition results, since during an aft, the unmeshed region is a collection of subdomains of this same simple type. If we allow the input domains to be connected, but not necessarily simply connected, the description is a bit more complex. For example, if the input domain, $\mathcal{D}$, is the annulus between two simple closed polygonal curves, C_0 and C_1, then the unmeshed region of an aft will contain a subdomain that is an annulus, until the aft forms a triangle with base edge, E, on one curve and opposite vertex, P, on the other (see Figure 1). Immediately after the creation of this triangle, all the connected components of the unmeshed region will be simply connected. But the component containing the node P will not be bounded by a simple closed curve; its bounding curve intersects itself at P.

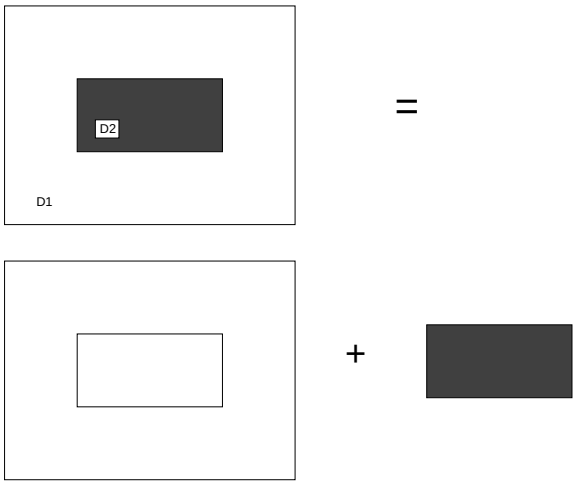

Figure 2: Geometry of modeling an inclusion region

Regions of finite element modeling interest are commonly collections of connected domains with disjoint interiors representing subregions of differing material properties. For example, Figure 2 shows a rectangular region with an outer annular subdomain, $D1$, of one material and an inner core rectangular subdomain, $D2$, of a second material. As input to an aft, this region would be presented as the pair of connected domains $D1$, $D2$, which must satisfy some consistency conditions on their common boundaries. Afts typically build meshes on each domain of such an input set independently, although not necessarily sequentially. Hence we can assume without loss of generality that the input to the polyalgorithm is a connected polygonal domain. We will now give a formal description of the boundary curves and frontal edge set that occurs in the meshing of such a domain by the polyalgorithm.

formal description

An edge, E, is a directed line segment between an origin node, $E.orign$, and a destination node, $E.destn$, i.e. an ordered pair of nodes. We will use $-E$ to denote the edge between the same nodes but with the opposite direction.

We now specify the required mathematical properties that will qualify a collection of edges to be a *frontal edge set*, (fes), in the balance of this paper. We will designate the frontal edge sets of our discussion by $\mathcal{C}$. By virtue of these defining properties, a fes has a basic geometric structure, i.e. its edges form the bounding polygonal curves of a disjoint collection of connected regions. The rationale for these definitions is that if an algorithm conforms to the polyalgorithm, the sequence of boundary edge updates that it makes maintains the collection of boundary edges as a frontal edge set. Thus, at each stage of the aft, we can identify the connected subdomains of the fes as the unmeshed regions of the input domain.

For a collection of edges to be a fes, we require that the edges not intersect except possibly at their end nodes, and we require each edge, E, to have a unique predecessor edge, $pred(E)$, in $\mathcal{C}$ i.e. $pred(E).destn = E.orign$, and a unique successor edge, $succ(E)$. However, an aft requires that more than two edges should be allowed to meet at a vertex, so we require further qualification of the uniqueness of predecessor and successor edges. For

this, we define the turning angle, $\theta(E,F)$, $-\pi < \theta < \pi$, between any two edges E and F for which $E.destn = F.orign$ as shown in Figure 3, positive in the counter clockwise direction. We require that the successor edge of E make the maximum turning angle of any edge, F,

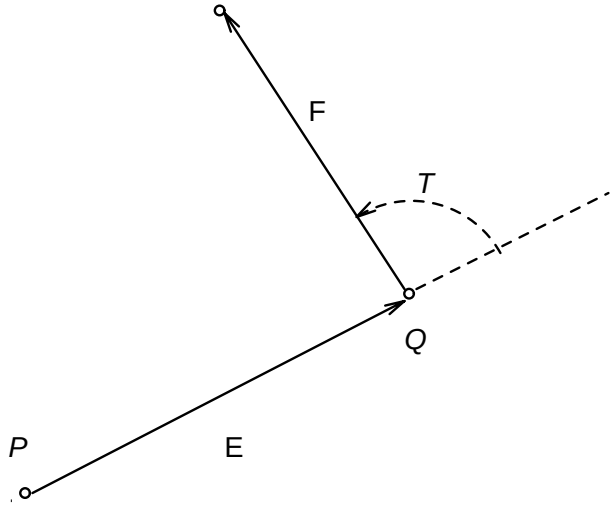

Figure 3: Turning angle from edge E to edge F

for which $F.orign = E.destn$. So, for example, in the case of edges E, V, U, W of Figure 4 in which all the edges have Q as an endpoint, and $E.destn = Q$, we identify the successor

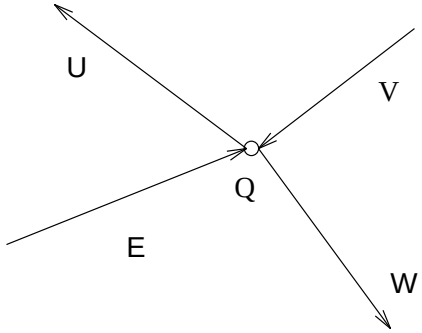

Figure 4: Multiple edges incident on node Q

of E to be the first edge on the left of E, e.g.

$$\theta(E, succ(E)) = \max_{G \in \{U,W\}} (\theta(E,G)). \tag{1}$$

This implies that $U = succ(E)$. Since W must have a unique predecessor, there must be an edge V with $V.destn = Q$ such that $W = succ(V)$ as shown. We can see that (1) and the uniqueness of predecessor and successor edges ensures that edges which are incoming to and outgoing from a single vertex such as Q of Figure 4 must alternate, with the unmeshed region contained between a clockwise sequential pair of incoming and outgoing edges.

It follows from these definitions that a frontal edge set, $\mathcal{C}$, can be partitioned into closed oriented polygonal curves which we will call the component curves of $\mathcal{C}$. In fact, these

component curves can be identified as the equivalence classes of edges in $\mathcal{C}$ determined by the equivalence relation defined between edges E and F in $\mathcal{C}$ if there is a chain of successor edges starting with E and ending with F. Although these curves can intersect themselves at isolated nodes, (1) ensures that they cannot cross themselves at such a node, so that their orientation is well defined. We identify the point set to the left of the curve as its interior, which may be bounded or unbounded. If the curve does not intersect itself, the Jordan curve theorem establishes that it divides the plane into two connected subdomains, and the bounded one of them is simply connected. If the curve does intersect itself at one or more isolated nodes, then a limiting argument involving circular arcs about these nodes from each edge to its successor can be used to ensure that the curve interiors continue to be connected, although the exteriors are no longer necessarily connected. We will now use *curve* to mean a closed polygonal curve of frontal edges, (i.e. an equivalence class of the frontal edge set). The curve that includes an edge E will be denoted $\mathcal{C}(E)$ and referred to as a component curve of $\mathcal{C}$.

Through these component curves, a frontal edge set defines the collection of subdomains of the region that remain to be meshed. If $\mathcal{C}(E)$ has a bounded interior which contains no other component curves of $\mathcal{C}$, it identifies one such connected subdomain that is simply connected. A description of these subdomains is complicated, however, by the possibility that they may have 'holes' in them, i.e. $\mathcal{C}(E)$ may contain m component curves $\mathcal{C}(F_k)$, for $k=$ 1,2,...m, with unbounded interiors (i.e. clockwise orientations). We can unify our notation by relabelling E as F_0. Then

$$\mathcal{S} = \bigcap_{k=0}^{m} interior\, \mathcal{C}(F_k) \tag{2}$$

is a multiply connected subdomain of the region defined by the fes. Each component curve of $\mathcal{C}$ of finite interior determines one such connected subdomain. We will refer to them as component subdomains of $\mathcal{C}$ and to their union as the interior of $\mathcal{C}$. When we wish to identify a component subdomain with an edge, G, on its boundary, we will designate it $\mathcal{S}(G)$. In the appendix, we establish a formula for the number of triangles in a mesh on $\mathcal{S}$ which is used in the next section to control the polyalgorithm.

We assume that the input to the polyalgorithm is some boundary description, $\mathcal{D}$, for a multiply connected domain. It is common in the aft methods to modify the edges of $\mathcal{D}$ to provide control over the sizes and shapes of triangles generated. We assume that this has already been done, i.e. a method conforming to the polyalgorithm will produce meshes with the input edges unmodified in the triangulation.

The polyalgorithm initializes its frontal edge set to represent this domain; it then maintains the fes as a description of the unmeshed regions and terminates when it is empty. The fact that the updating of the fes by the polyalgorithm maintains its defining properties is essential to the argument that algorithms which conform to the polyalgorithm are correct afts. The choice of data structure that an aft algorithm employs to represent the fes is an important efficiency consideration. The polyalgorithm has been written without specifying a particular representation for a fes, $\mathcal{C}$, and the minimal requirements of the subtasks refer to searching $\mathcal{C}$ for edges, or nodes, with specific properties. Some of the accompanying comments mention efficiencies possible if some particular representation of the component curves, or subregions is available. In §3, the use of the constrained Delaunay triangulation to

represent the connected component subregions, $\mathcal{S}(E)$, provides an representation of $\mathcal{C}$ that maintains its geometry explicitly and a corresponding measure of efficiency.

2.2 The polyalgorithm

In Figure 5, we show the top level decomposition of the polyalgorithm for aft methods as a procedure named **Basic_aft**. The substeps are given as procedures and functions that are described subsequently . These procedures appear to have actual argument lists, with arguments typed in **Basic_aft**. The lists have the form:

<input variables> ; <output variables>

with updated data appearing twice. In fact, however, most of the data for the algorithm can be regarded as global, and each procedure is used once to perform a task on these data, except for **Visible** and **Add_triangle**. We have included this typing and parameter listing as an aid to understanding the task of each procedure.

The algorithm input consists of the original edge set $\mathcal{D}$ and a parameter, fin, which will ensure that a finite triangulation is generated. There are two fundamentally different strategies for this which are described in the minimal requirements for the function **Mesh_size_constraint** below. The polyalgorithm is organized so that exactly one triangle is added to the triangulation $\mathcal{T}$ for each pass through the while loop.

We now describe the minimal requirements for each subtask, as well as commenting on possible, or typical strategies for meeting these requirements.

Convert_to_frontal_edge_set($\mathcal{D}$; $\mathcal{C}$)

minimal requirement - A frontal edge set is constructed from $\mathcal{D}$ such that the region to be meshed is contained in its interior and this fes is assigned to $\mathcal{C}$.

Get_next_edge($\mathcal{C}$; E)

minimal requirement - Any edge of $\mathcal{C}$ is returned, which will be referred to as the current edge.

The next triangle to be generated will have this current edge as a side. This subtask specifies an ordering strategy for processing the edges in $\mathcal{C}$, such as longest or shortest first cf. Tilch, [27].

Two alternate candidate vertices for the third vertex of a triangle to be constructed on the current edge are now located.

Compute_new_candidate(E ; R)

minimal requirement - This subtask may return any point of the plane on the left of the current edge.

```
Procedure Basic_aft(𝒟, fin)
type   node                  R, S
       edge                  E
       logical               New, New_is_preferred, Mesh_size_constraint, Visible
       control parameter     fin
       boundary edge set     𝒟
       frontal edge set      𝒞
Convert_to_frontal_edge_set(𝒟 ; 𝒞)
while 𝒞 ≠ ∅
   Get_next_edge(𝒞 ; E)
   Compute_new_candidate( E ; R )
   Compute_existing_candidate( E, 𝒞 ; S )
   New ← New_is_preferred(R, S) and not Mesh_size_constraint(fin)
   if New
      then
         New ← Visible(E, 𝒞, R)
   endif
   if New
      then
         Update_internal(𝒞,E, R, fin ; 𝒞,fin)
         Add_triangle(𝒯, E, R ; 𝒯)
      else
         Update_boundary(𝒞, E, S, fin ; 𝒞, fin)
         Add_triangle(𝒯, E, S ; 𝒯)
   endif
endwhile
```

Figure 5: Basic Advancing Front Technique Polyalgorithm

Compute_existing_candidate(E, $\mathcal{C}$; S)

minimal requirement - A vertex, S, from $\mathcal{C}$ is returned which is visible from the current edge.

A simple geometric argument can be used to assure that there is at least one node of $\mathcal{C}$ visible from E. Although elementary, this observation is crucial to the correctness claim that each subtask of the polyalgorithm can be accomplished. This subtask may require the logical function, **Visible**, discussed below.

The triangles of the mesh serve a variety of perhaps competing goals; error control, desired stiffness matrix properties, as well as meeting the geometric constraints that they tile the domain, $\mathcal{D}$ (see discussions in [2] or [25]). The candidate triangle based on R is typically selected to serve the goals that are independent of the geometric constraints; i.e. R is typically an 'ideal' choice of vertex selected independently of $\mathcal{C}$. It is not clear at this

stage of the polyalgorithm that this triangle is feasible, i.e. that it lies entirely in the interior of $\mathcal{S}(E)$. Even if it is feasible, its use may force edges into $\mathcal{C}$ for which it is difficult to form satisfactory triangles subsequently. The candidate vertex S is a feasible default alternative to R from the vertices of $\mathcal{S}(E)$ which could create a triangle that takes account of the geometry of the domain being meshed, typically serving the nongeometric goals of triangle shape as well as possible.

logical function New_is_preferred(R, S)

minimal requirement - **New_is_preferred** must be assigned $false$ if R and S are the same node.

Otherwise, **New_is_preferred** can be assigned arbitrarily in the polyalgorithm. In the extreme case that **New_is_preferred** is always set $false$, the corresponding aft method becomes a form of greedy algorithm for computing a triangulation of $\mathcal{D}$ [1].

The logical function, **New_is_preferred**, embodies the strategy of a preference for one of these candidates. As incorporated in the polyalgorithm, this decision is made assuming that both triangles are feasible, and then if a preference for R is established, its feasibility is checked by the function **Visible**. This organization is based on the assumption that checking the visibility is computationally expensive compared to the preference decision, which is typically the case.

logical function Mesh_size_constraint(fin)

minimal requirement - This function must ensure that only a finite number of new vertices are generated.

In mesh generation algorithms generally, there are two approaches to ensuring that a finite number of triangles are generated. One is to directly control the number of triangles, and the other is to control some aspect of the triangles which ensures termination of the method accepting whatever number of triangles results. For each of these approaches, we give an example of how the minimal requirement for **Mesh_size_constraint**(fin) could be met.

For the first approach, a maximum number of triangles permitted in the final mesh is specified. We need to be able to predict the minimum number of triangles required to complete the triangulation of the interior of a fes. It is well known that the minimal triangulation of a connected finite domain with m holes and $m+1$ simple closed boundary curves having a total of E_b edges is

$$N_{min} = E_b + 2(m-1) \tag{3}$$

using only the boundary vertices,(e.g. Fuhring, [12].) In the appendix, we show that this formula holds for the minimal triangulation of a component subdomain of a fes (the $\mathcal{S}$ of (2)), even though the boundary curves (the $\mathcal{C}(F_i)$ of (2)) may not be simple. Since the edges of different component subdomains are distinct, we can conclude that the minimal triangulation of the interior of a fes is also given by (3), where E_b is the total number of

[1] See §6.2.1 of [22].

edges in the fes and m is the sum of the number of holes in all component subdomains of the fes.

To use this to limit the number of triangles produced by an aft, we require the control parameter, fin, to be an array of four integers carrying the information:

$N(\mathcal{T})$ = the number of triangles currently in the triangulation $\mathcal{T}$.

$N(\mathcal{C})$ = the number of edges in $\mathcal{C}$, the current fes.

m = the sum of the number of holes in the interior of $\mathcal{C}$

N_{max} = the maximum allowable number of triangles in the final mesh.

We can ensure that the polyalgorithm will not produce more than N_{max} triangles if we set:

$$\begin{aligned} \text{Mesh\_size\_constraint} &= true \text{ if } N(\mathcal{T}) + N(\mathcal{C}) + 2(m-1) \geq N_{max} \\ &= false \text{ otherwise} \end{aligned} \tag{4}$$

The numbers $N(\mathcal{T})$, $N(\mathcal{C})$ and m can be updated in the **update** operations of the polyalgorithm. However, we note that an aft conforming to the polyalgorithm produces a sequence of fes , $\mathcal{C}^j$ for which m_j is non-increasing, starting with m_0 = the number of 'holes' in the original domain $\mathcal{D}$. Consequently, a practical simplification of (4) would be to use for m the number of holes in the original domain.

In the second approach, we require the control parameter, fin, to be an array of two positive real numbers. One of these is a minimum area to be permitted for a triangle to be formed by introducing a new node, A_{min}. The second is the area, $A(E,R)$, of the triangle defined by the current edge E, and the new candidate node, R. If we set

$$\begin{aligned} \text{Mesh\_size\_constraint} &= true \text{ if } A(E,R) < A_{min} \\ &= false \text{ otherwise} \end{aligned} \tag{5}$$

then only a finite number of new vertices will be created.

logical function Visible(E, $\mathcal{C}$, U)

minimal requirement - **Visible** must be assigned *true* if the line segments from each point on edge E do not intersect the boundary curve(s) of $\mathcal{S}(E)$, *false* otherwise.

This function is used with $U = R$ ensure that a new candidate node, R, is feasible for the creation of a new triangle.

Update_internal($\mathcal{C}$,E, R, fin ; $\mathcal{C}$,fin)

minimal requirement - This subtask must insert new edges F from $E.orign$ to R and G from R to $E.destn$ into $\mathcal{C}$. It deletes E from $\mathcal{C}$ and updates the control parameter, *fin*.

We note that the insertion of F and G, plus the deletion of E maintains the connectivity of $\mathcal{S}(E)$ and it results in a net increase of one edge in $\mathcal{C}(E)$.

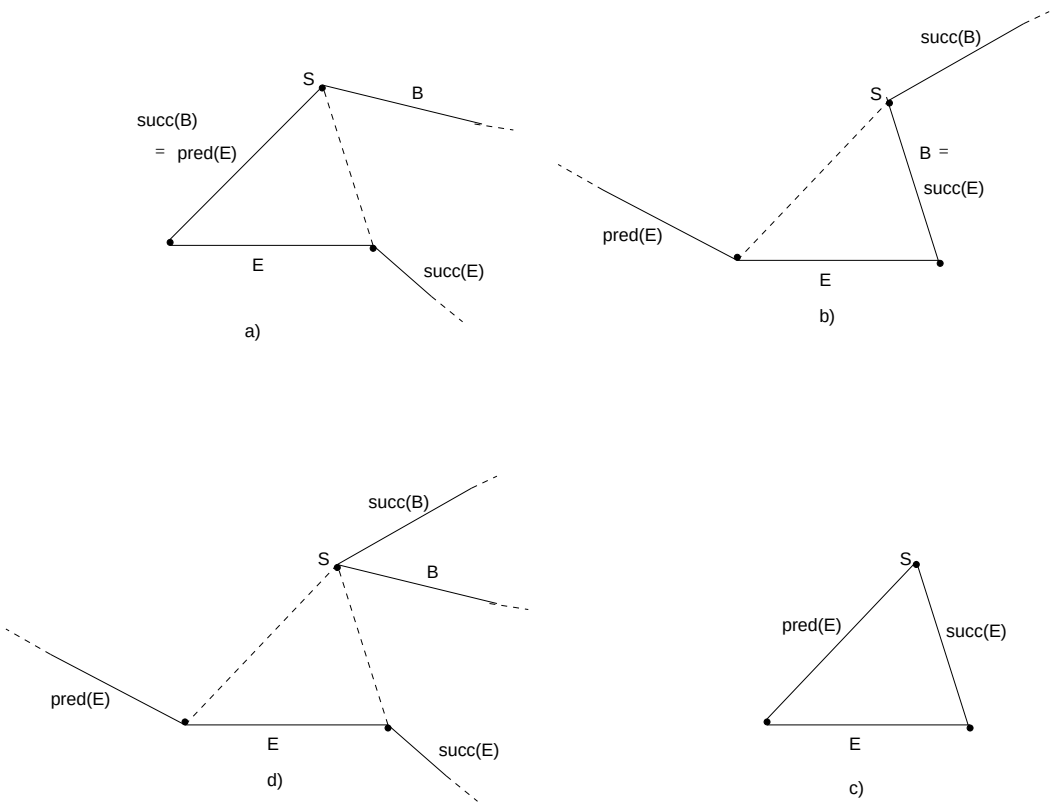

Figure 6: Updating the boundary of $\mathcal{C}(E)$ for existing candidate node S - four cases

Update_boundary$(\mathcal{C}, E, S, fin\ ;\ \mathcal{C}, fin)$

minimal requirement - This subtask must update fin and make modifications to $\mathcal{C}$ that are described in terms of new edges F from $E.orign$ to S and G from S to $E.destn$.

- $if\ -F\ \in\ \mathcal{C}\ then$ delete $-F$ from $\mathcal{C}$ $else$ insert F in $\mathcal{C}$
- $if\ -G\ \in\ \mathcal{C}\ then$ delete $-G$ from $\mathcal{C}$ $else$ insert G in $\mathcal{C}$
- delete E from $\mathcal{C}$

To see that this update maintains $\mathcal{C}$ as a frontal edge set, we note from Figure 6 that there are basically three different cases that occur. In the figure, edge B is the edge of $\mathcal{S}(E)$ with $B.destn\ =\ S$. The first case is illustrated by subfigures marked a) and b) in Figure 6 and which illustrate $-F \in \mathcal{C}$ and $-G \in \mathcal{C}$ respectively. The connectivity of the subdomain $\mathcal{S}(E)$ remains unchanged and $\mathcal{C}(E)$ undergoes a net reduction of 1 edge. The second case is marked c); in this case $\mathcal{C}(E)$ simply consists of three edges. They form the next triangle and the effect of the update is to remove $\mathcal{C}(E)$ from $\mathcal{C}$. The third case is illustrated by d) of Figure(6). The effect on $\mathcal{S}(E)$ depends on whether S lies on $\mathcal{C}(E)$, in which case $\mathcal{S}(E)$ is subdivided into two new subdomains, or $\mathcal{S}(E)$ is multiply connected, and S lies on $\mathcal{C}(B) \neq \mathcal{C}(E)$. In this latter subcase, after the update, $\mathcal{C}(E)$ will intersect itself at S, and $\mathcal{S}(E)$ remains a connected subdomain but its connectivity (i.e. number of 'holes') is reduced by one. In both versions of this third case, the number of edges in $\mathcal{C}$ increases by one.

The statement of the minimal requirements for this update is correct and succinct, but it may not reflect the typical efficiency considerations of aft algorithms adequately. A direct interpretation might suggest that efficiency would be served by having a rapid and inexpensive way to determine whether a given edge is present in $\mathcal{C}$, or not. However, the use of the special geometric context of this question in this update can be used to avoid requiring a general mechanism. In the next section, we comment on how a particular data representation of the subregions $\mathcal{S}(E)$ can be used for this purpose.

Add_triangle$((\mathcal{T}, E, U\ ;\ \mathcal{T})$

minimal requirement - The triangle formed by edge E and opposite vertex U is added to $\mathcal{T}$.

The correctness of the polyalgorithm follows from making the following observations:

- each subtask can be successfully executed for its minimal requirements
- the properties of the frontal edge set are maintained
- the *while* loop terminates.

3 Using a constrained Delaunay triangulation for $\mathcal{S}(E)$

A key efficiency issue in the performance of an aft is the representation of the frontal edge set, $\mathcal{C}$. The representation must support insertions, deletions, and searching in the sense of point location and checking visibility. List oriented data structures from computational geometry for these tasks have been discussed by Dannelongue and Tanguy, [10] and by Tilch, [26]. In this section, we propose that an efficient, high level representation for the unmeshed region in an aft algorithm is the use of its constrained Delaunay triangulation. I.e. each connected component, $\mathcal{S}(E)$, would be represented by its constrained Delaunay triangulation. We can describe the constrained Delaunay triangulation of $\mathcal{S}(E)$ as a triangulation with the defining property that if any node is contained in the interior of the circumcircle of a triangle, then every interior point of the triangle is separated from this node by a boundary edge of $\mathcal{S}(E)$. We will shorten "the constrained Delaunay triangulation of $\mathcal{S}(E)$" to cd$(\mathcal{S}(E))$ in the sequel. Although, strictly speaking, cd$(\mathcal{S}(E))$ is a collection of triangles, we will refer to an edge of cd$(\mathcal{S}(E))$ meaning an edge of one of the triangles.

The efficiency of using a triangulation to represent $\mathcal{S}(E)$ for visibility checking has been noted by Chew [8], by Guibas et al [14], by Lo [18], and by Müller, Roe, and Deconinck [20]. The use of the constrained Delaunay triangulation has some additional efficiency advantages if there are mesh quality goals of avoiding small angles in the generated mesh, or other reasons why the resulting mesh should be the constrained Delaunay triangulation of its vertices plus the region's boundary and interface edges. We will refer to mesh generation with these quality goals as isotropic mesh generation, (Simpson, [25]).

In §2, we saw that during an aft algorithm's execution, the domain, $\mathcal{D}$, is partitioned into the unmeshed region $= \cup_{E\in\mathcal{C}}\ \mathcal{S}(E)$, and the subregion which has already been triangulated, $\mathcal{D} - \cup_{E\in\mathcal{C}}\ \mathcal{S}(E)$, with the frontal edge set, $\mathcal{C}$, as the boundary between the two

subregions. In the preceding paragraph, we proposed that there are significant efficiencies to be gained by representing the unmeshed subregion by its constrained Delaunay triangulation. For isotropic mesh generation, it is appropriate to extend this proposal to maintaining the meshed subregion also as the constrained Delaunay triangulation of its nodes, the required edges of the input domain $\mathcal{D}$, and the frontal edge set, $\mathcal{C}$. Since $\mathcal{C}$ is part of the required edge set for both these subregions, it can be seen that the extended proposal is equivalent to proposing that aft methods for isotropic mesh generation maintain at each stage the constrained Delaunay triangulation of the total configuration of nodes, boundary edges and frontal edges that it is processing. We have introduced this idea via the two subregions of $\mathcal{D}$, the meshed and unmeshed regions, because the motivations for each are different. The major motivation in the case of the unmeshed region is efficiency, while the major motivation in the case of the meshed region is the quality goal of isotropic mesh generation.

We discuss the implications of this representation for the subtasks of the polyalgorithm of Figure 5 that are affected by it. We assume that a data structure for $\mathrm{cd}(\mathcal{S}(E))$ is used that allows easy access from a boundary edge to the triangle incident on it, and from one triangle to its neighbour sharing an edge in common.

Convert_to_frontal_edge_set($\mathcal{D}$; $\mathcal{C}$)

In addition to constructing $\mathcal{C}$ from $\mathcal{D}$, this subtask requires the identification of the components $\mathcal{S}(E)$ of $\mathcal{D}$, and the construction of $\mathrm{cd}(\mathcal{S}(E))$ for each. Several algorithms for computing the constrained Delaunay triangulation appear in the literature, e.g. Chew, [7], or Lee and Lin, [17]. The primary source for this study, however, is Cline and Renka, [9] which is supported by a publically available implementation.

Compute_existing_candidate(E , $\mathcal{S}(E)$; S)

If we select S to be the node of the triangle of $\mathrm{cd}(\mathcal{S}(E))$ incident on E, we can be sure that S is visible from E. If the goal is isotropic mesh generation, then this is an appropriate heuristic for selecting S, which can ensure that the resultant triangulation is the constrained Delaunay triangulation of its vertices if combined with edge swapping in $\mathcal{T}$ as discussed below under **Add_triangle**. If some other choice of S is used to pursue other mesh quality goals then $\mathrm{cd}(\mathcal{S}(E))$ can be used in checking the visibility of S as discussed below.

New_is_preferred(R, S)

A useful criterion for isotropic mesh generation is to prefer the new candidate node, R, if the triangle formed by E and R is in $\mathrm{cd}(\mathcal{S}(E) \cup R)$. Let us assume that the existing candidate node, S, is selected to be the vertex of the triangle of $\mathrm{cd}(\mathcal{S}(E))$ with base E as suggested above. Then an efficiency of using $\mathrm{cd}(\mathcal{S}(E))$ arises from the fact that it is sufficient to check that S lies outside the circumcircle of the triangle formed by E and R. The justification for this claim of efficiency is provided in the following lemma. In it we will designate the triangle formed by E and a node V by $T(E,V)$ and the circumcircle of $T(E,V)$ by $C(E,V)$.

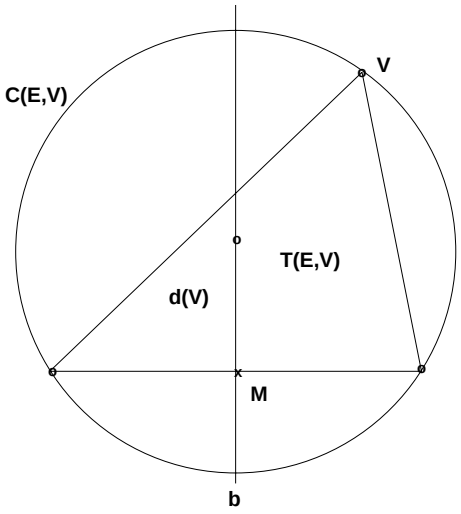

Figure 7: Configuration of T(E,V) , C(E,V) for lemma 3.1

Lemma 3.1 *If R is visible from E, and $T(E,S) \in \mathrm{cd}(\mathcal{S}(E))$ and E, R, S are not cocircular, then*

$$T(E,R) \in \mathrm{cd}(\mathcal{S}(E) \cup R) \Leftrightarrow S \notin C(E,R)$$

Proof Let M be the midpoint of E and b be the perpendicular bisector of E. Then, for any node V, the center of $C(E,V)$ lies on b. Let the distance from M to this point on b be $d(V)$, measured positively if the center lies to the left of E as illustrated in Figure 7.

It is easy to see that

$$V \in C(E,W) \Leftrightarrow \mathrm{d}(V) < \mathrm{d}(W)$$

and E, V and W are cocircular if and only if $\mathrm{d}(V) = \mathrm{d}(W)$. Hence, if E, R, and S are not cocircular then either $\mathrm{d}(R) > \mathrm{d}(S)$ or $\mathrm{d}(R) < \mathrm{d}(S)$. In the first case, $T(E,S)$ continues to satisfy the empty circle criterion when R is added to $\mathrm{cd}(\mathcal{S}(E))$, so $T(E,S) \in \mathrm{cd}(\mathcal{S}(E) \cup R)$ and consequently $T(E,R) \notin \mathrm{cd}(\mathcal{S}(E) \cup R)$

In the second case, we note that for any node V of $\mathcal{S}(E)$ visible from $T(E,S)$

$$\mathrm{d}(V) \geq \mathrm{d}(S) > \mathrm{d}(R)$$

and hence no edge of $\mathcal{S}(E)$ can terminate at a visible node inside either $C(E,R)$ or $C(E,S)$. To conclude that if $\mathrm{d}(R) < \mathrm{d}(S)$ then $T(E,R) \in \mathrm{cd}(\mathcal{S}(E) \cup R)$, we must check that either $C(E,R)$ is empty, or if $V \in C(E,R)$ then V is separated from $T(E,R)$ by the boundary of $\mathcal{S}(E)$. If $V \in \mathcal{S}(E)$ is in $C(E,R)$, then $V \in C(E,S)$, and hence V is separated from $T(E,S)$ by a boundary edge, F of $\mathcal{S}(E)$, which cannot have its endpoints in $C(E,S)$. Since the endpoints of F cannot lie in $C(E,S)$, $T(E,R)$ must lie on the same side of F as $T(E,S)$. Consequently, V is separated from $T(E,R)$ by F and $T(E,R)$ satisfies the empty circle criterion for $\mathrm{cd}(\mathcal{S}(E) \cup R)$.

logical function Visible(E,R)

For the visibility subtask of the polyalgorithm, with the $\mathrm{cd}(\mathcal{S}(E))$ triangulation, and candidate node R preferred according to the discussion of **New_is_preferred**, we can make

```
Procedure Check_visible(E, R, cd(S(E));Visible, Left, Right)
type   node                      R, M
       edge                      E, F, G
       triangle                  T
       constr. Del. triangul.    cd(S(E))
       logical                   Visible
       edge_list                 Left, Right
Visible ← true
M ← (E.orgn + E.destn)/2
G ← E
T ← select triangle, given G,
                  where triangle ∈ cd(S(E)) and has edge G
while R ∉ T and Visible
    F ← select edge given R,M,T where edge ∈ T and RM cuts edge
    if F ∈ S(E)
       then     Visible ← false
       else
          if F.destn = G.orgn
             then
                add F to Right
             else
                add F to Left
          M ← compute intersection point of RM and F
          G ← −F
          T ← select triangle, given G,
                           where triangle ∈ cd(S(E)) and has edge G
endwhile
```

Figure 8: Visibility check using triangulation of cd($\mathcal{S}(E)$)

the following specific discussion. In this case, we can conclude that there are no nodes in the circumcircle of E and R that are visible from E. Consequently, if R is not visible from any one point on E, there must be a boundary edge,H , of $\mathcal{S}(E)$ that passes through this circumcircle, and consequently, R is hidden from every point on E. So it suffices to check that R is visible from one point of E, which we choose to be the midpoint, M. In Figure 8, we give a description of an algorithm to check this visibility; the SQL like syntax of this description is discussed in [24]. In this algorithm, two queues of edges of cd($\mathcal{S}(E)$) that cut the line segment M to R are created which will be used by **Update_internal** if R is visible. Since we now are returning these two lists to the polyalgorithm, to be conveyed to the update subtask, we change **Visible** from being a function to being a procedure, named **Check_visible**. The operation

$F \leftarrow$ *select* edge *given* R, G, T *where* edge $\in T$ *and* RM *cuts* edge

selects the edge of triangle T which the line of sight MR cuts if it leaves T. Note that this operation does not fail since there are no visible vertices of $\mathrm{cd}(\mathcal{S}(E))$ in the circumcircle of the triangle formed by E and R (see **New_is_preferred**).

Update_internal($\mathcal{C}$,E, R, fin ; $\mathcal{C}$,fin)), **Update_boundary**($\mathcal{C}, E, S, fin$; $\mathcal{C}, fin$)

With the use of $\cup_{E\in\mathcal{C}}\mathrm{cd}(\mathcal{S}(E))$ to represent $\mathcal{C}$, the insertions of new edges, F and G, referred to in §2.2 involve the retriangulation of a subset of $\mathrm{cd}(\mathcal{S}(E))$. In theorem 1 of Cline and Renka, [9], a specification for this subset is provided , and several algorithms are presented for support of the retriangulation. This process could, in principle, be carried out sequentially for edges F and G, but the theorem and techniques of [9] can be trivially extended to show that if T^e is the set of triangles whose interiors intersect either edge F or edge G, and B^e is the boundary of $\bigcup\ T^e$, then it is sufficient to construct $\mathrm{cd}(B^e \bigcup F \bigcup G)$.

In the case of **Update_boundary**($\mathcal{C}, E, S, fin$; $\mathcal{C}, fin$), the test $-F \in \mathcal{C}$? becomes simply whether F is an edge of the triangle of base E and opposite vertex S in $\mathrm{cd}(\mathcal{S}(E))$.

Add-triangle($\mathcal{T}, E, U$; $\mathcal{T}$)

Under the updates just described , the edges of $\mathcal{C}$ partition $\mathcal{D}$ into the unmeshed subregion, $\cup_{E\in\mathcal{C}}\mathcal{S}(E)$, which is triangulated by $\cup_{E\in\mathcal{C}}\ \mathrm{cd}(\mathcal{S}(E))$ and the meshed region, $\mathcal{D} - \cup_{E\in\mathcal{C}}\mathcal{S}(E)$, which is triangulated by $\mathcal{T}$. We are not , however, assured the $\mathcal{T}$ is the constrained Delaunay triangulation of its vertices with regard to edges of $\mathcal{C}$ and $\mathcal{D}$.

We now designate the meshed region as $\mathcal{D}(meshed)$ and discuss the minimal requirement for **Add-triangle** of ensuring that $\mathcal{T}$ is maintained as the constrained Delaunay triangulation of $\mathcal{D}(meshed)$. We return to the notation of Lemma 3.1 in which $T(E,U)$ designates the triangle formed by edge E and opposite vertex U. Let us further designate the meshed regions before and after the addition of $T(E,U)$ by $\mathcal{D}_{old}(meshed)$ and $\mathcal{D}_{new}(meshed)$, and their triangulations as $\mathcal{T}_{old}$ and $\mathcal{T}_{new}$. If E is an edge of the constrained Delaunay triangulation of $\mathcal{D}_{new}(meshed)$, then the minimal requirement of **Add-triangle** remains to simply add $T(E,U)$ to the data structure for $\mathcal{T}_{old}$. However, if not, then a series of edge swaps as described in [9] will be necessary to obtain $\mathcal{T}_{new}$ from $\mathcal{T}_{old}$.

A test for whether E is in $\mathcal{T}_{new}$ can be described by an extension of Lemma 3.1. Let $T(-E,\hat{S})$ be the triangle with edge $-E$ in $\mathcal{T}_{old}$. If $\hat{S} \notin C(E,U)$, or $\hat{S}$ is not visible from $T(E,U)$, then $T(E,U)$ is in the constrained Delaunay triangulation of $\mathcal{D}_{new}(meshed)$.

4 Conclusion

Practical unstructured FEM mesh generation requires attention to issues of

- conformance to geometric constraints
- performance efficiency.
- implications of triangle shapes for properties of the global stiffness matrix
- control of discretization errors in the computed solutions

The polyalgorithm of §2 addresses the first of these issues by providing a framework such that meeting the minimal requirements of each step assures that a mesh conforming to the geometry is generated. The flexibility that remains unspecified by this polyalgorithm can then be directed to other issues of this list.

During the execution of an aft algorithm, the unmeshed region is the union of disjoint, connected subdomains and the representation of these subdomains has an important influence on performance efficiency, the second of the issues in the above list. The constrained Delaunay triangulation is an effective high level data structure for representing the individual subdomains. In §3, we discussed this observation and the extensions of the polyalgorithm that can be made to incorporate it.

It is natural to contemplate how this framework for aft methods can be extended to generating three dimensional tetrahedral meshes for polytopes with two dimensional triangulated surfaces. However, we can quickly demonstrate that the polyalgorithm of §2 cannot be extended directly to three dimensions. As mentioned in the discussion of the logical function **New_is_preferred**, if this function is simply set to *false* for all inputs, then the polyalgorithm generates a triangulation of the input domain using its existing vertices. A direct extension of it should have the same property. But, we know that there are polytopes with triangulated surfaces that cannot be meshed by tetrahedra using existing vertices only, e.g. Ruppert and Seidel, [23]. It is easy to see, using the examples of [23], that the attempt to extend the subtask **Compute_existing_candidate** to three dimensions can fail.

References

[1] F Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23:345–405, 1991.

[2] M Bern and D Eppstein. Mesh generation and optimal triangulation. In F K Huang, editor, *Computing in Euclidean Geometry*. World Scientific, 1992.

[3] C Borgers. Generalized Delaunay triangulations of non-convex domains. *Computers and Math Applns*, 20:45–49, 1990.

[4] T D Bui and V N Hanh. Automatic mesh generation for finite element analysis. *Computing*, 44:305–329, 1990.

[5] A Bykat. Automatic generation of triangular grid:i-subdivision of a general polygon into convex subregions.ii - triangulation of convex polygons. *Intl J for Num Meth in Engrg*, 10:1329–1342, 1976.

[6] J Cabello, R Lohner, and O-P Jacquotte. A variational method for the optimization of directionally stretched elements generated by the advancing front method. In A S Arcilla, J Hauser, P R Eiseman, and J F Thompson, editors, *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, pages 521–532. Elsevier Science Pub, North Holland, 1991.

[7] L P Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:205–219, 1989.

[8] L P Chew. Guaranteed-quality mesh generation for curved surfaces. In *9th Annual Symposium on Comp Geometry*, pages 274–280, San Diego, California, 1993. ACM.

[9] A K Cline and R J Renka. A constrained two-dimensional triangulation and the solution of closest node problems in the presence of barriers. *SIAM J of Num Anal*, 27:1305–1321, 1990.

[10] H H Dannelongue and P A Tanguy. Efficient data structures for adaptive remeshing with the FEM. *JCP*, 91:94–109, 1990.

[11] S Farestam. A geometry based approach to the generation of unstructured surface grids. In P.-J. Laurent, A. LeMehaute, and L. Schumaker, editors, *Curves and Surfaces II*. AKPeters, Boston, 1994.

[12] H Fuhring. The application of node-element rules for forecasting problems in the generation of finite element meshes. *Inter J of Numer methods in Engrg.*, 21:617–629, 1975.

[13] P L George. *Automatic Mesh Generation : application to finite element methods*. John Wiley and Sons, Paris : Masson, 1991.

[14] L Guibas, J Hershberger, D Leven, M Sharir, and R E Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[15] C M Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann, San Mateo, Calif, 1989.

[16] Z Jian-Ming, S Ke-Ran, Z Ke-Ding, and Z Qiong-Hua. Computing constrained triangulation and Delaunay triangulation: A new algorithm. *IEEE Trans on Magnetics*, 26:694–697, 1990.

[17] D T Lee and A Lin. Generalized Delaunay triangulation for planar graphs. *Disc and Comp Geom*, 1:201–217, 1986.

[18] S H Lo. Delaunay triangulation of non-convex planar domains. *Intl J for Num Methods in Engrg*, 28:2695–2707, 1989.

[19] R Lohner and P Parikh. Generation of three-dimensional unstructured grids by the advancing front method. *Intl J for Num Methods in Fluids*, 8:1135–1149, 1988.

[20] J D Müller, P L Roe, and H Deconinck. A frontal approach for node generation in Delaunay triangulations. In *Unstructured Grid Methods for Advection Dominated Flows*, number R-787. AGARD, 1992. presented in an AGARD-FDP-VKI Special Course Unstructured Grid Methods for Advection Dominated Flows at the VKI 2-6 March 1992 and at NASA Ames, 28 Sep - 2 Oct 1992.

[21] J Peraire, M Vahdati, K Morgan, and O C Zienkiewicz. Adaptive remeshing for compressible flow computations. *JCP*, 72, 1987.

[22] F P Preparata and M I Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[23] J Ruppert and R Seidel. On the difficulty of tretrahedralizing 3-dimensional non- convex polyhedra. *Proc Fifth Annual Symposium on Comp Geometry*, pages 380–392, 1989.

[24] R B Simpson. A data base abstraction for unstructured triangular mesh algorithms. Technical Report CS-92-13, Dept of Computer Sci, U of Waterloo, Waterloo, Ontario, Canada , N2L 3G1, March 1992.

[25] R B Simpson. Anisotropic mesh transformations and optimal error control. *Applied Num Math*, 14:to appear, 1994.

[26] R Tilch. Unstructured grids, adaptive remeshing and mesh generation for Navier-stokes. In *12th Intern'l Conf. on Numerical Methods in Fluid Dynamics.* Oxford University, July 1990.

[27] R Tilch. *Unstructured Grids for the Compressible Navier-Stokes Equations.* PhD thesis, CERFACS, 31057, Toulouse, France, 1990.

[28] M-G Vallet. *Génération de maillages éléments finis anisotropes et adaptatifs.* PhD thesis, Paris 6, 1992.

[29] M G Vallet, F Hecht, and B Mantel. Anisotropic control of mesh generation based on a Vornoi type method. In A S Arcilla, J Hauser, P R Eiseman, and J F Thompson, editors, *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, pages 93–104. Elsevier Science Pub, North Holland, 1991.

Appendix - How many triangles in a mesh on the interior of a frontal edge set?

In §2, we require a formula for the minimal number of triangles required to triangulate the interior of a fes. The interior of a fes comprises the interiors of its separate connected subdomains, $\mathcal{S}$, with boundary curves $\mathcal{C}(F_k)$ for $k = 0$ to m, as described in (2). If the boundary curves of $\mathcal{S}$ are all simple closed curves, then there is a standard formula for the number of triangles in a mesh on the interior of $\mathcal{S}$. If we let T be the number of triangles, E_b the number of edges on the boundary, and V_b and V_i be the number of vertices on the boundary and in the interior of $\mathcal{S}$ respectively, then $E_b = V_b$ and

$$T = E_b + 2V_i + 2(m-1) \tag{6}$$

(e.g. Fuhring page 619 , [12].)

For a component subdomain of a fes, the boundary curves are oriented and closed , but need not be simple. The standard argument for (6) breaks down for non simple boundary curves because we no longer have $E_b = V_b$. The following lemma states that formula (6) holds for a component of a fes, nevertheless. We note that in the context of an aft method the interiors of the component subdomains of a fes are the *unmeshed* parts of the region $\mathcal{D}$, so they normally are considered to have no interior vertices.

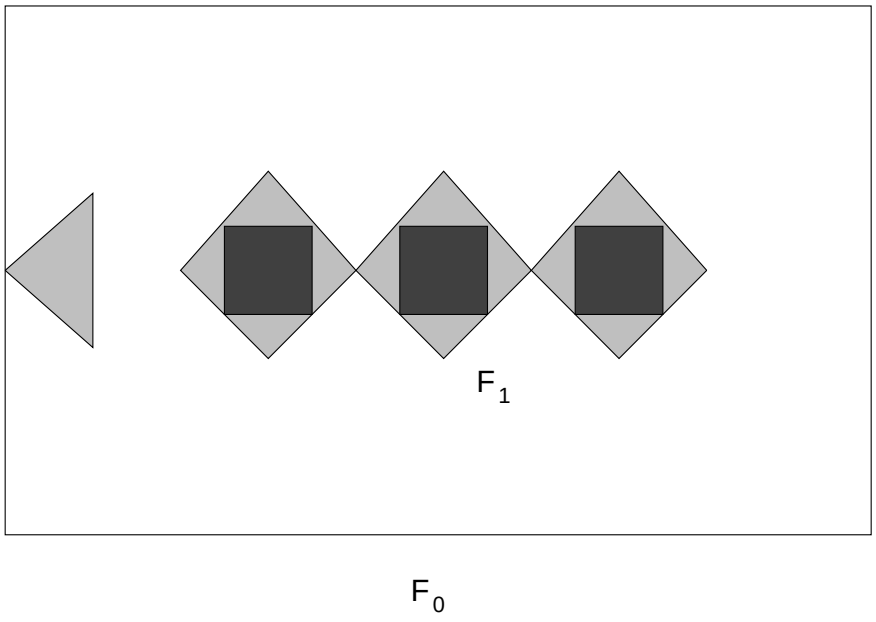

Figure 9: Example of a component subdomain of a frontal edge set

Lemma Let $\mathcal{S}$ be a component subdomain of a frontal edge set. The number of triangles in a triangulation of $\mathcal{S}$ is given by (6).

Proof: A basic starting point for formula relating the number of components in a mesh is the Euler-Poincare formula for planar graphs

$$f + v - e = 2 \tag{7}$$

where f is the number of faces in the graph, v the number of vertices and e the number of edges (e.g. Hoffman, [15], § 2.3.3)

To apply this to a triangulation of the interior of $\mathcal{S}$, we must specialize (7) to triangular meshes. It is well known that for a triangular mesh that

$$e = E_b + E_i = \frac{3}{2}T + \frac{1}{2}E_b \tag{8}$$

where E_i is the number of edges internal to the mesh, ([12]).

In this context, the faces of the graph interior to $\mathcal{S}$ are triangles of the mesh and the faces exterior to $\mathcal{S}$ are the simply connected subdomains of finite area in the exteriors of the $\mathcal{C}(F_k)$ plus the one infinite subdomain exterior to $\mathcal{C}(F_0)$. Let us designate the number of simply connected subdomains of finite area in the exterior of $\mathcal{C}(F_k)$ by H_k for $k = 0$ to m. If $\mathcal{C}(F_k)$ is a simple closed curve, then $H_k = 1$ for $k \geq 1$ and $H_0 = 0$ by the Jordan curve theorem. In Figure 9, we show an example where $m = 1$, $H_0 = 1$ and $H_1 = 3$. The simply connected subdomains of finite area in the exterior of the $\mathcal{C}(F_k)$ are shaded. The single lightly shaded triangle at the left is the face of finite area in the exterior of $\mathcal{C}(F_0)$. The regions of mixed light and dark shading in the centre of the figure are the faces determined by $\mathcal{C}(F_1)$. The darker shaded portions of these regions are three 'holes' in the original domain $\mathcal{D}$. As this example illustrates, each face of the exterior of $\mathcal{C}(F_k)$ for $k \geq 1$ contains one or more holes of the

original domain, plus possibly some triangles external to the fes; however, this substructure is irrelevant to our current derivation. So

$$f = T + \sum_{k=0}^{m} H_k + 1 \ ; \ \ v = V_b + V_i \tag{9}$$

If we substitute (8) and (9) into (7), we get

$$T = 2V_b - E_b + 2V_i + 2\sum_{k=0}^{m} H_k - 2 \tag{10}$$

This equation has not taken account of the connection between V_b, H_k and E_b. In the common case of a domain $\mathcal{S}$ with simple closed curves for all its boundaries, we have

$$H_0 = 0, H_k = 1, V_b = E_b$$

For a component subdomain with non simple boundary curves, these relations are replaced by

$$E_b = V_b + \sum_{k=0}^{m} H_k - m \tag{11}$$

To establish (11), consider the following scan of the edges of the boundary curves of the component subdomain. We accumulate the number of vertices in s_1 and $\sum_{k=0}^{m} H_k - m$ in s_2.

(Initializations)

$m(P) = 0$ for all P in $\mathcal{S}$ (marker for counting visits to vertex P)

$H_0 \leftarrow 0$ (because simply connected $\mathcal{C}(F_0)$ has no exterior face of finite area)

for $k = 1$ to m

  $H_k \leftarrow 1$ (because simply connected $\mathcal{C}(F_k)$ has one exterior face of finite area)

$s_1 = 0 \ ; \ s_2 = 0$

(Edge scan)

for $k = 0$ to m

  for $E \in \mathcal{C}(F_k)$

    $P \leftarrow E.destn$

    $m(P) \leftarrow$ m(P)+1

    if $m(P) = 1$

      then $s_1 \leftarrow s_1 + 1$

      else $s_2 \leftarrow s_2 + 1$

We note that as each edge is scanned either s_1 , or s_2 is incremented. The first time a vertex is visited s_1 is incremented. Each subsequent time a vertex is visited, the current face of the exterior of $\mathcal{C}(F_k)$ is closed, and a new face is initiated, hence s_2 is incremented.

Now, we can use (11) to replace $\sum_{k=0}^{1} H_k$ in (10) to get (6). Note that since the variables in (6) are all additive for an ensemble of component subdomains, this formula also holds for the fes itself, if the variables refer to the subdomain ensemble totals.